smxUSBH™

USB Host Stack

***smxUSBH* is a simple USB host stack for embedded systems. It is written in C, and can be ported to any hardware platform. *smxUSBH* is optimized for SMX®, but can be ported to other RTOSs or run stand alone. It is modularized so that only what is needed will be linked into the final application.**

Layers

- **Class Driver Layer** provides USB device support such as mouse, keyboard, hub, printer, mass storage, and serial.
- **USB Driver Layer (USBD), or Core,** provides the common USB device functionality.
- **Host Controller Driver (HCD) Layer** provides host controller driver functionality and contains root hub support.
- **Porting Layer** provides service functions related to the hardware, OS, and compiler.

Code Size

Code size can vary greatly depending upon the processor, compiler, and optimization level. The code sizes, below, are for Intel x86, Microsoft C, optimization level O2.

Component	Size (KB)
Core (USBD, Port. Layer, hub driver)	19
EHCI Driver	17
OHCI Driver	15.3
UHCI Driver	17.6
ISP116x Driver	8
ISP1362 Driver	8.5
ISP176x Driver	16.5
Mass Storage Device Driver	10
Mouse and Keyboard Device Driver	5
Printer Device Driver	3

RISC processors require about 20% more code space.

Features

- Class drivers are available for mass storage drives, mice, keyboards, printers, hubs, and serial devices.
- Compatibility with ARM, ColdFire, PowerPC, x86, and other CPU's.
- Cascading hub support for up to 127 devices.
- Compliant with USB Specification 2.0.
- Compliant with EHCI 1.0, OHCI 1.0a, and UHCI 1.1 Specifications.
- Philips ISP1161/0, ISP1362, and ISP176x support.
- Atmel AT91 and Freescale ColdFire USB host controller support.
- Supports all four USB data transfers (control, bulk, isochronous, and interrupt).
- Written entirely in ANSI-C.
- Porting, integration, and development services are available.
- Typical code footprint is less than 50 KB for a CISC processor.
- Optimized for SMX® RTOS.
- Integrated with *smxFile* and *smxFS* for USB disk support.
- OHCI and UHCI support in real mode and DOS.

Data Size

All RAM that *smxUSBH* uses for data is pre-allocated from the heap when *smxUSBH* is first initialized. Following is a table of RAM usage:

Host Controller	Core + Mass Storage Only	Core + All Device Drivers
EHCI	16KB	20KB
OHCI	16KB	20KB
UHCI	76KB	80KB
ISP116x	8KB	12KB
ISP1362	6KB	8KB
ISP176x	8KB	12KB

UHCI requires much more memory than OHCI because the hardware is more rudimentary and the software must do more work. The UHCI RAM requirements include 1024 *Transfer Descriptors* (*TDs*) of 64 bytes, each (64KB total). The number of TDs can be reduced, but performance suffers. For example with 128 TDs, performance is reduced by a factor of 10. For OHCI, there is no RAM vs. performance tradeoff. OHCI is obviously preferable to UHCI for limited RAM systems.

Performance for Mass Storage

The following table shows **raw** transfer speed from and to a USB flash disk. 30MB total transfers are done 4KB at a time.

Host Controller	Raw Reading	Raw Writing
EHCI (NEC)	10825 KB/sec	7831 KB/sec
OHCI (NEC)	667 KB/sec	540 KB/sec
UHCI (VIA)	434 KB/sec	416 KB/sec
ISP116x (Philips)	340 KB/sec	335 KB/sec
ISP1362 (Philips)	472 KB/s	455 KB/s
ISP176x (Philips)	7425 KB/s	3214 KB/s

The following table shows ***smxFS*** read/write performance for the same USB flash disk. Total file size is 30MB with 4KB transferred, at a time.

Host Controller	File Read	File Write
EHCI (NEC)	10556 KB/sec	7211 KB/sec
OHCI (NEC)	651 KB/sec	523 KB/sec
UHCI (VIA)	429 KB/sec	408 KB/sec
ISP116x (Philips)	332 KB/sec	329 KB/sec
ISP1362 (Philips)	469 KB/s	454 KB/s
ISP176x (Philips)	7023 KB/s	3072 KB/s

Class Driver API

Mass Storage
su_MStorIO(buf_ptr, first_sector, num_sectors,
reading)
su_MstorMediaInserted()
su_MstorMediaRemoved()
su_MStorSectorNum()
su_MStorSectorSize()

Mouse
su_MouseInit()
su_MouseInserted()
su_MouseRelease()
su_MouseSetCallback(handler)

Keyboard
su_KbdInit()
su_KbdInserted()
su_KbdRelease()
su_KbdSetCallback(handler)

Printer
su_PrnID(pdata, len)
su_PrnInit()
su_PrnInserted()
su_PrnRead(pdata, len)
su_PrnRelease()
su_PrnReset()
su_PrnStatus()
su_PrnWrite(pdata, len)

Serial
su_SerialOpen(id)
su_SerialClose(id)
su_SerialInserted(id)
su_SerialRead(id, pdata, len)
su_SerialWrite(id, pdata, len)
su_SerialGetLineState(port, pstate)
su_SerialGetLineCoding(port, rate, parity,
databits, stopbits)

Writing a New Class Driver

smxUSBH provides a class driver template and a section in the manual to help you write a new class driver.

Serial Class Driver

The serial driver supports any device that Windows XP or 2000 can support without a custom driver. Unfortunately, most serial adapters do require installation of a driver on Windows. Additional code must be developed to support such an adapter, which could require significant effort. Please see the smxUSBH User's Guide for details, and discuss your requirements with us.

Real Mode and DOS Support

Although the focus of SMX products is on modern embedded processors such as ARM and ColdFire, we recognize the need for legacy x86 systems to add USB support, especially for flash disks. Because of this, we have put effort into supporting OHCI and UHCI in real mode.

OHCI required some effort to support because it uses memory mapped I/O and the PCI BIOS assigns a high address near the top of the 4GB memory space, which is not accessible in real mode. We provide two solutions for this. In one case a 386 or better is required. See the OHCI section of the smxUSBH User's Guide for details.

UHCI was fairly easy to support since it uses x86 I/O space for access to UHCI registers. All that is required is a 386 or better because it requires 32-bit I/O instructions.